Imitation-based Artificial Player for Pic-A-Boo

Elio Freitas
Instituto Superior Técnico
Lisboa, Portugal
elio.freitas@tecnico.ulisboa.pt

***Abstract*—Pic-a-boo is game about taking pictures in a dark room. The game-play maintains most information hidden from the player requiring guessing to win. Pic-a-boo is a game that requires more than one player to play, and at the time it did not provide any type artificial player to fill the missing players. Creating an agent with hidden information is challenging. To solve this problem this project created an agent capable of imitating a human through the use of a neural network. The agent trained by playing against human players. The final agent was able to navigate the map believably, however other actions like taking photos were reproduced randomly without any discernible human pattern.**

***Index Terms*—Game; Artificial Intelligence; Human Behavior; Agent; Neural Networks; Imitation; Training;**

I. Introduction

It is challenging for an Artificial Intelligence (AI) to imitate human behavior. The challenge lies in the difficulty of predicting human behavior. There are many humans with different personalities. This make it hard to express human behavior as a simple mathematical model or algorithm. However, there have been several attempts at achieving this. Some of the most successful ones used recordings of players to train algorithms. Others analyzed common strategies used in games and reproduced them.

Creating a **human-like** AI is the current challenge that the game Pic-a-boo is facing. A **human-like** AI is an AI capable of imitating the actions patterns of a human. Pic-a-boo is about taking pictures of your enemies in a dark room. The winner is the player able to take pictures of all the other players. Pic-a-boo uses a top-down view map with several dark zones, the players are only able to only see their enemies and themselves in light sources. Pic-a-boo is a game that can be classified as an action game, this type of games require fast response times. This limits the type of solutions to be implemented because the decisions must be in real time. Otherwise, the credibility of the agents is compromised. But why should an AI imitate a human player?. Several reasons exist, One reason is to fill vacant players for multiplayer games, because sometimes finding 4 human players is not feasible so an AI that plays the part helps in obtaining the complete game experience. Another one is to present a fair challenge to a new player. A match between humans with the same game playing condition its perceived as more fair, compared to a match between an AI that only seeks to win the game. This type of AI will be designated as perfect-play AI for this project. Perfect-play AI is an AI that seeks to solve the game. Perfect-play AI seeks the best answer to the problem presented in the game. For instance, in real time strategy games like Starcraft [1], players are sometimes required to manage each unit individually. A perfect play AI could manage each unit almost instantaneously. For an average player this would be impossible. A perfect-play AI is not limited by fairness and may be frustrating for newer players. New players haven't learned the game yet. A novice confronting a perfect player is an unfair challenge. Even for an expert in the game, whom may like their challenge, it is still unfair. It is unfair because it is rare for a human to perform in a perfect and instantaneous manner. It is unlikely that humans will be able to manage units at the same speed as a computer. Instead, a **human-like** AI purpose is to imitate the way a human will play, this includes imitating mistakes. But what requirements exist for creating a **human-like** AI for Pic-a-boo? First, it is necessary to find an appropriate algorithm. Given that a key factor in selecting an algorithm is the response times using an artificial neural network (ANN) might be appropriate. Training a neural network can take from a few minutes to several hours. However, once trained a ANN does not take long times to compute, so it won't compromise the response time of the game.

To create a **human-like** AI another important aspect of the development is to capture the player actions. This is because the data is going to be the basis for the player model that the AI is going to imitate. Finally, the resulting agent similarity to a human requires measuring. The measuring will help to determine if the resulting agent is behaving like a human.

II. Related Work

A. Player Modelling

Player modelling is the study of computational models of players in games. It includes the detection, modelling, prediction, and expression of human player characteristics. These characteristics manifest themselves through cognitive, affective and behavioral patterns. The main goal of player modelling research is to understand how players interact with a game. Following this, a model is produced based upon that interaction [1].

While playing video games humans produce dynamic and complex behaviors. There are 2 approaches to model these complex behaviors when creating player models. The first approach is a theory based approach which relies on making a rough approximation of the human's behavior. The agents

[1]StarCraft II: Wings of Liberty real-time strategy video game developed and published by Blizzard Entertainment in 2010.

produced from this method will behave in the predetermined behavior or strategy, for example fleeing attackers. According to [2] we can call this model-based approach a **persona** or personality. The second approach is a data-driven approach which rely on gathering data from players to analyze the patterns inside the data. The agents produced from this method will then behave in a way that agrees with the patterns extracted from the play traces. We can call this data-driven approach a **clone**.

Both cases have problems, for instance, clones are prone to over-fitting, where the agent only learns to cope with situations appearing in the play traces. In the case of new situations, the agent might have irregular behavior. Personalities to a large extent solve the irregular behavior problem. The solution provided is through learning a robust general strategy. This solution is also called a decision-making style. But a decision-making style can be general and predictable.

This project requires choosing between clones or personas. Trying to formulate a general strategy to imitate a player is difficult and will lead to a predictable agent. Guesswork is one of the main components of Pic-a-boo's matches so being predictable would eliminate a big part of the challenge. The cloning method might bring irregular behavior but humans are not a regular themselves so it isn't a big issue. So between the two the cloning methodology is the chosen.

B. Artificial Neural Networks

Artificial Neural networks (ANN) are a family of algorithms used to simulate human capabilities among other uses, for instance image recognition, face identification and self driving cars [3]. These tasks are easy for humans to perform but not for a computer. This is because it's not easy to describe these task in a couple of reproducible steps. It is not easy because the numbers of steps might be too big or have too many caveats, thus making it impractical to code. ANN overcome this problem by imitating a human brain.

There are various examples of using an ANN for creating human behavior, for instance the company [2] behind Candy Crush Saga[3]. The company sought to improve their game testing by using a Convolutional neural network. CNN is a specific type of ANN. CNN original purpose was for image-like data feature extraction.

The need came from the speed at which they wanted to release new content. The speed of release was not aligned with the velocity of the testing efforts. This because humans were required to test the product. Human testing comes with the disadvantage of introducing latency and costs [4]. They saw an advantage in creating a human behaving algorithm to automate testing. The idea was that an agent could run tests at any time after the new levels were completed. These automated tests would help find problems in the new levels increasing the speed of development and reducing the costs of human testers.

[2]King Digital Entertainment

[3]Candy crush saga video demo: https://www.youtube.com/watch?v=d5Rf0AnjEg

According to [4] there have been many attempts at creating a human agent, for example simulating a player playing a level using an agent with a simple heuristic. But these approaches lack one important aspect. The approaches were trying to imitate humans without human input. For this fact the researchers sought to include human data because they thought it could be beneficial.

The general approach they used was an agent that would play the game with the help of a CNN to guide decisions. The agent would then generate the metric of interest needed to test the quality of the levels. Then the researchers would compare the values from the agent with the values from humans. The closer the strategy of the agent to that of human players the better.

Researchers compared the approach against another approach using Monte Carlo Search Tree. The decision to benchmarks with an MCTS agent came out of pragmatism. It was practical to use because the company already had a working MCTS agent. So it was useful to compare the two to check if the new approach was better than the previous one. The conclusion was that the new approach outperformed the MCTS agent. The CNN agent had more predictive power, execution efficiency and also outperformed humans. Human play testers took around 7 days to complete a level. Instead, the new CNN agent could perform the task in less than a minute which is a huge optimization of time.

Another example of using an ANN for creating a human agent came from the researchers from the University of Essex. Their goal was to create a general game playing AI. They used a CNN paired with a Q-learning algorithm. This solution is called Deep-Q-Network which is a combination between a CNN and Q-learning. The CNN would analyze the image translate it into inputs and make actions. While the Q-learning functioned as a training supervisor for the CNN, Q-learning would analyze the state-action relationship to select what data to feed into the CNN to train, based on previous states.

The researchers decided to use screen capture to receive the inputs of the games, which is a **sub-symbolic** input source. Meaning it is made by constituent entities that are not direct representations of the game state, for example pixels. In contrast to using **symbolic** inputs, which are a representation system in which the atomic constituents are a direct representation of the game state. For example the system recording the player position [5].

The decision came because humans receive most of their information through visual sensors. The reception of the inputs from a visual source is beneficial for a general game playing AI. Because this allows decoupling the algorithm from the game software, in turn increasing its adaptability to other problems.

The method used contains two steps. The first step was to process the image, they shrank each defined block down into one pixel. Then normalized the RGB value and expanded into the smallest size allowed. The modification of the image normalized the inputs. The normalization need came because the CNN structure is the same for all games. Also, the

algorithms could get overloaded with the whole information contained in the image.

For the second step the Q-learning algorithm consumes the image as a current state analyzing the reward. The Q-learning will choose the next action to train the CNN based on the Q values. The CNN then receives the input image and generates an action.

Results were positive, the agent was able to learn how to solve both static [4] and stochastic [5] games. The accumulative winning percentage in static games increased the more the agent played. The same applied to the cumulative average score in stochastic games. Suggesting the agent was applying knowledge acquired during the previous plays.

In general after checking these works it seems feasible to use an algorithm from the ANN family. Candy crush is a game played in real time proving that ANN based solutions are capable of fast response times. Regarding the work done at the university of Essex, they demonstrate the ability of a neural network to play not only one game but several. Even more in a learning style close to a human because it is based on vision. While the goal was different, because they wanted to achieve a general perfect-play agent and this work seeks to create a human-like agent.

One of the main decision lies in how the neural network will receive data. As previously mentioned there are 2 approaches to the inputs **symbolic** or **sub-symbolic**.

The main problem of using a neural network is the training phase because this process requires an adequate quantity of data. The quantity of data is difficult to assess but is related to how many scenarios the ANN requires learning. The problem is there is no available data already created like for the candy crush saga example. Data collection requires recording several play-through of the game. These recording are time-consuming.

A **sub-symbolic** approach like screen capturing brings the need of analyzing, compressing and simplifying the video. Otherwise, the amount of data to input into the ANN would be too much hampering the training times. Also, Pic-a-boo is a game played with fog of war. Which does not let you see your character or the enemy character. This means it is an imperfect information game. So a big chunk of information represented in the video will be useless to train requiring more refining efforts.

This fact shift the balance to favor using **symbolic** inputs. A symbolic data model has as much information as required. This means more focused data to train ANN and less refining efforts which translates into faster training times.

C. Unity Machine Learning Agents Toolkit

As the previous examples have shown, ANN has seen some advances and usages in recent years. Simulation platforms are one of the many tools that had help in the advancement of ANN and AIs in general. Examples of these platforms are Arcade Learning Environment (ALE) [6], VizDoom [7] and Mujoco [8]. The existence of these environments have been essential because they provide the means to benchmark and test algorithms. Also, these simulation platforms serve not only to enable algorithmic improvements, but also as a starting point for training models which may be deployed in the real world. A prime example of this is the work being done to train autonomous robots within a simulator, and transfer that model to a real-world robot [9]. In these cases, the simulator provides a safe, controlled, and accelerated training environment.

Pic-a-boo is a game that was developed using the Unity game engine and conveniently Unity provides a good platform for developing AIs. This platform is known as Unity Machine Learning Agents Toolkit (UMLAT). UMLAT is an open source project that enables researchers and developers to create simulated environments using the Unity Editor and interact with them using a Python API. The toolkit is built to take full advantage of the properties of the Unity Engine which makes it a strong research platform [10].

UMLAT provides many resources necessary for creating simulated environments in Unity. It contains two components. First a Software development kit (SDK), which contains all functionality required to create environments within the Unity Editor and associated C# scripts. Second a Python package which enables interfacing from outside of Unity with environments built using the SDK.

UMLAT presents a high-level Application Program Interface (API) for creating AIs using ANNs. This means that designing and implementing an ANN is done through configuring a series of parameters exposed by the API. For instance parameters like: number of layers, numbers of units in layers, input format, etc. UMLAT will read these parameters and create the desired modification in the provided basic ANN architectures. To create these ANNs UMLAT uses the TensorFlow python package. TensorFlow is an open-source software from Google used for data-flow and differentiable programming. It is mostly a math library, and one of its main uses is machine learning applications such as neural networks [11].

UMLAT provides a set of baseline algorithms implemented with TensorFlow, which can be used for starting the development of novel algorithms. The toolkit currently provides an implementation of Proximal Policy Optimization (PPO) [12], a state-of-the-art Deep Reinforcement Learning algorithm, with the option to extend it using an Intrinsic Curiosity Module (ICM) [13] and Long Short Term Memory (LSTM) [14]. However, the most relevant of the algorithms for this project is the Behavioral Cloning, a simple Imitation Learning algorithm [15] that was implemented to showcase the Unity Editor as a tool for recording expert demonstrations.

D. Measuring Artificial Intelligence

1) Human Tests: One important aspect of this work will be how to test the final results. There is no standard way of measuring human behavior. One good way that has been used

[4]Game in which a single decision is made by each player, and each player has no knowledge of the decision made by the other players

[5]Game where each player selects an action and receives a payoff, the games moves to a new random state based on the previous state and actions

before [16] is to ask a human because humans have the innate ability to recognize other humans. In the previous mentioned work [16] the researchers used this method. The researchers presented a video of the resulting agent to various people. The video contained several play-traces of the agents mixed with players. Each participant had to identify between humans and agents.

This test has a close relationship with the Turing test [17] which consist of an imitation game. The idea of the test is that a machine has to try to pretend to be a human. The machine should answer questions put to it and will only pass if it manages to convince a human that the machine is a human. This human should not have a considerable knowledge of how machine works [17]. By using this kind of tests we can get an idea of the agent quality. However, the test will always be a little biased, because it will depend on the previous knowledge of the participants. Some participants may have knowledge of the game. This knowledge helps to identify quirks in the agent that a normal player will not perform. However, a less expert person may not identify the agents. So previous knowledge is an important factor for these tests.

2) Confusion Matrix: A more unbiased way of measuring an AI will be using a mathematical model. In [18] the main tool used for evaluating agents is the **confusion matrix**. For [18] evaluating a model is the process of understanding how good are the predictions the model makes. A **confusion matrix** is a table of rows and columns that presents predictions versus actual outcomes. It is used to understand how it is performing based on giving the correct answer at the right moment. binary confusion matrix measures 4 values. To better understand this, imagine having an AI capable of classifying between humans or an AI acting as human the values will be: True positive (TP), False positive (FP), True Negative (TN), False Negative (FN).

With these values in hand a more detailed analysis on the performance of the model can be made by calculating the following four relationships shown in 1.

RECALL	TP/ (TP + FN)
PRECISION	TP / (TP + FP)
F1SCORE	(TP + TN) / (TOTAL)
ACCURACY	2TP / (2TP + FP + FN)

Fig. 1. Binary confusion matrix relationships

.

3) Multi Class Confusion Matrix: The goal of a great number of ANNs models is to learn to map a given input to a corresponding known output. For **prediction** tasks, the output comes in the form of a single label. For **regression** tasks, it is a single value. ANNs are able to solve these simple single-output problems like binary classification with ease. For instance ANNs are able to filter spam in an email system or label images.

However, these single output models are not coping well with the increasing needs of today's complex decision-making. As a result, there is a pressing need for new paradigms. Here, multi-output learning has emerged as a solution. The aim is to simultaneously predict multiple outputs given an input, which means it is possible to solve far more complex decision-making problems.

Pic-a-boo player actions can't be solved with a binary output alone. For instance moving in the game requires 5 different outputs up, down, left, right and neutral. The problem lies in how these outputs are to be measured. A binary confusion matrix is not enough to solve the problem. To solve the measuring problem this work is going to use a multi-class confusion matrix [19].

Binary classification problems usually focus on a positive and a negative class which must be detected. Instead, in a multi-class classification problem, we need to categorize outputs into 1 of N different classes. For example, we can use an AI trained to predict player movements between left, right or remain neutral. A multi-class confusion matrix will then measure n by n values for this case n=3 that would mean 9 measures, corresponding to the combinations of the possibilities: AA, AB, AC, BA, BB, BC, CA, CB, CC.

With these values an analysis similar to the binary confusion matrix can be made by calculating 3 of the previous relationships as can be seen in 2.

	RECALL	PRECISION	F1SCORE
A	AA/(AA+BA+CA)	AA/(AA+AB+AC)	(2*(AP * AR))/(AP + AR)
B	BB/(AB+BB+CB)	BB/(BA+BB+BC)	(2*(BP * BR))/(BP + BR)
C	CC/(AC+BC+CC)	CC/(CA+CB+CC)	(2*(CP * CR))/(CP + CR)

Fig. 2. Multi confusion matrix metrics

.

After calculating each one of these we can them calculate the mean recall precision and mean f1-score for the model as seen in 3.

Mean Recall	(AP + BP + CP)/ 3
Mean Precision	(AR + BR + CR)/ 3
Mean F1-Score	(AF1S + BF1S + CF1S)/ 3

Fig. 3. Multi Confusion matrix mean metrics

.

These values are also known as macro-precision, macro-recall and macro-F1-Score, in this work they will be known as the macro-metrics of a confusion matrix. Nonetheless, there is a second set of metrics that can be calculated. These metrics are what we are going to call the micro-metrics, the micro-metrics do not analyze the matrix by class but instead as a whole. However, there is a particularity, in a multi-class confusion matrix when analyzing the whole matrix a TP and TN have the same meaning both are correct predictions. FP and FN also have the same meaning both are incorrect predictions so that means that the micro-metrics are all equivalent to the accuracy which can be seen in 4.

In addition to theses four parameters calculated from the confusion matrix, other metrics exist for instance the hamming

ACCURACY	(AA + BB + CC)/ (AA + AB + AC + BA + BB + BC + CA + CB + CC)

Fig. 4. Multi Confusion accuracy

.

loss [20], commonly used with strings. The hamming loss computes the average difference between the predicted and actual output. These differences are calculated by counting the number of different outputs in a set. Now going back to the previous example of an AI trained to predict player movements between left, right or remain neutral, imagine we are comparing two sequences of actions:

- Player: "left, right, left, left, right, neutral, neutral"
- AI prediction: "left, right, right, right, right, neutral, neutral"

Between these sequences there are 2 differences. D will be the sum of all the differences found and N the number of comparisons.

$$HL = D/N$$

So for this example, the hamming loss will be:

$$HL = 2/7 = 0.28$$

Normally hamming loss is used for string however the output of the model can be treated as a string, so hamming loss can be applied.

Another related metric that can be calculated is the percentage of perfect predictions of the model. Meaning that the difference between both actions is 0, For this project this metric will be called **Perfect match**. Z will be the number of zero matches and N will be the total number of sequences.

$$PM = Z/N$$

III. Pic-a-boo Game concept

Pic-a-boo is a game whose main objective is taking pictures of enemy players. The game is played in a top-down view map which can be seen in 5 [6]. The screen is shared between all players so each player has the same information as their adversaries. Players may move in horizontal, vertical and diagonal directions.

Taking pictures of the player is done by using the flash action near an enemy. Being photographed does not mean you are out of the game. All enemies must be captured to win the game, repeated photos don't count. Taking a photo inside the game means first navigating near the position of your target. Next, with the avatar near the target the player must press the flash button action. The flashlight direction cannot be controlled independently it will point at the last direction that the player moved. The flashlight is represented by a cone of light being emitted from the player avatar. Capturing a target

[6]Gameplay videos can be found at https://www.youtube.com/watch?v=cvScIMoUR-E

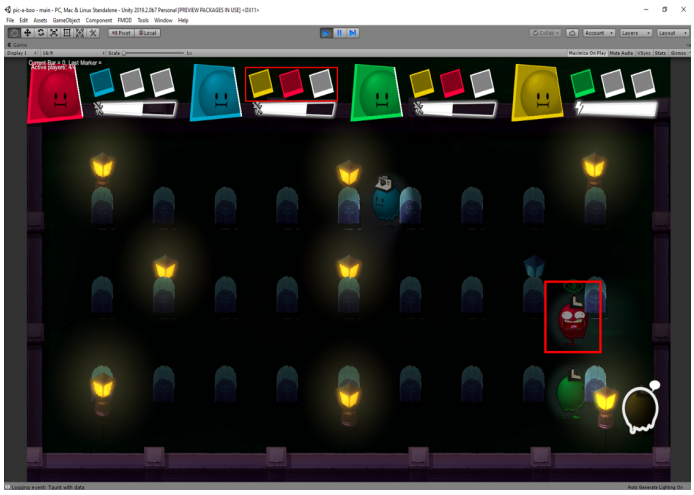

Fig. 5. Blue player current photograph.

with the initial flash will cause the target to enter a flashed state. In this flashed state the enemy can only move but may not perform any other action.

Given the fact that the map is dark the players can only see themselves near the light emitted by the lamps or after using the flash action. In addition, the player may also use the taunt action to see their player avatar without a light source and to lure the other players.

A. Game Actions

The game may be played with either a keyboard or a controller. In game, players can perform 6 actions which are:

- LEFT: Starts moving the player's avatar horizontally to the left.
- RIGHT: Starts moving the player's avatar horizontally to the right.
- UP: Starts moving the player's avatar vertically to the top.
- DOWN: Starts moving the player's avatar vertically to the bottom.
- FLASH: Turns on the player flashlight.
- TAUNT: Highlights the player showing its current position.

The controller button scheme may vary depending on the controller. The vertical and movement actions can be combined to move in a diagonal. For instance if the player is moving UP and combines it with LEFT the avatar starts moving in a 45 degree diagonal to the north-west. The avatar's velocity is constant so it can go from 0 to maximum speed with just one action. Colliding with an obstacle stops the player's avatar, colliding with an enemy causes the avatar to starts pushing the enemy in the same direction the player's avatar is going. The enemy may stop the push by walking in a direction opposite to where it is being pushed.

Flashing allows capturing enemies in a photo, however flashing has a cool-down period of 3 seconds. Following the use of the flash action the bar near the player's portrait will

begin to refill itself representing the cool-down period. The player should be able to use flash again once the bar is refilled.

IV. Implementation

A. Training environment architecture

The architecture used in this project is based on UMLAT. This project architecture is centered around three entities namely: **agent**, **brain**, and **academy**. The **academy** is the entity in charge of orchestrating the environment. This means that it controls the flow of each training session alongside with resetting each agent to its original state which mean configuring the agents positions, cool-downs and velocities to their initial values. The **agent** entity is in charge of controlling the different playable avatars while collecting observations and receiving actions from the **brain**. **Agents** collect information about the current game state in the form of observations and send them to the **brain**. Following this the **brain** sends actions to the **agent** which must be executed in the environment, Given the fact that the **agent** entity was in charge of executing actions, it was attached to the player avatar. The player avatar object is the object that the player uses to interact with the game, the player avatar was copied from Pic-a-boo preserving the same physics properties. There were 4 instances of this object running one for each player, 1 human and 3 agents. The **brain** was the one in charge of receiving the current state of the game in the form of observations and pass these values through the ANN to obtain a prediction.

B. Neural network architecture

UMLAT provides a high-level API for creating an ANN with Tensorflow. This means that to use and train an ANN from the toolkit it wasn't necessary to define nor implement any of the common steps like activation or optimization functions. Instead, the only requirements were to define the inputs and the topology. An overview of the architecture can be found in 6.

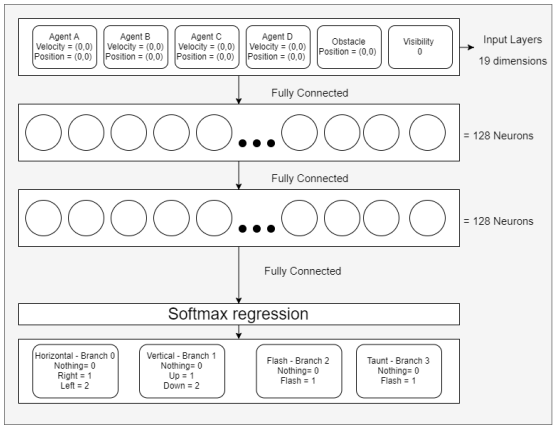

Fig. 6. Feed-forward Neural Network Topology.

The inputs are represented by the previously mentioned observations that were collected by the **agents** which are also related to the vector observation size defined in the **brain**. In the **brain** this value was defined as 19 and it can be divided into 3 categories which are:

- Position and velocity of each agent meaning two 2-Dimensional vectors for four agents, which adds to sixteen values.
- Position of the last collided object, again a 2-Dimensional vector which adds to two values.
- Visibility of the agent, a boolean, which adds to one value.

Following this, the next step was to define the network topology. This is done through the training configuration files provided by UMLAT in [21]. Parameters like hidden units and number of layers can be set in these files. Most parameters were left with the same values as default parameters provided by UMLAT except for **Max_Steps**. The **Max_Steps** value was lowered to 1.0e4 from 1.0e5 because after testing further training didn't improve the **agents**.

C. Neural network training

The training process of the ANN is handled entirely by UMLAT. UMLAT provided two type online training and offline training. To train the ANN this project used the online training mode. In this mode a human must actively play the game against the learning agents. This mode contrast with the offline training mode in the sense that training offline meant recording the game and feeding the results to the ANN.

After setting the training type, how UMLAT handles the training is not documented. However, the code is open-source so it was reverse engineered. The training is done using a python environment with the TensorFlow library. First the parameters defined in the configuration files are received and the topology of the ANN is created. Second the observations stream composed of the previously mentioned positions and velocities reaches the ANN where the stream is multiplied by the weights-matrix and summed by the bias.

```
weight_in = inputs * weights + bias
outputs = linearActivation(weight_in)
```

The values then pass through a linear activation function which controls when the neurons are active. This process is repeated as many times as there are layers. For this project the process was repeated two times because there were only two layers. The final step to obtain the predictions is to pass through a soft max function. The soft max function is going to normalize the result and perform a logistic regression. Logistic regression are used to predict categorical results like the ones in Pic-a-boo, for instance flashing and not flashing [22].

With the prediction finally obtained, the value is then compared to the human **brain** action. From this comparison the loss value is obtained and then the value is passed through an Adam optimizer.

The Adam optimizer is a variant of a stochastic gradient descent function, these functions are used to calculate a local minimum by approximating the gradient descent instead of calculating it with the whole data. This estimation is done by estimating it with part of the data. These functions are

commonly used as an optimization algorithm to reduces loss and updates the network weights based on the training data [23].

The process described above is repeated continuously in a training session, so each time the weights are updated, the prediction of the ANN changes to be more similar to the real action of the human because the Adam optimizer reduces the difference between both values.

D. Agent quality tests

To test the agents, it was necessary to enter an input to the ANN and get a prediction. The inputs of the ANN are the observations extracted from the game-state as mentioned before. So it was required to save these observations to later input them into the ANN. A small modification was done in the **agent** entity for it to record a player match into a JSON file. The JSON file contained the various observations of the game space through each frame alongside with what actions the human performed based on that observation. The values that the JSON file contained were:

- player: contains an object describing the player position and velocity.
- targets: contains an array objects describing the targets position and velocity.
- isVisible: a flag describing if the player avatar was visible.
- lastObstacleX: describe the horizontal position of the last collision with an object.
- lastObstacleY: describe the horizontal position of the last collision with an object
- horizontalMovement: The output horizontal action made by the player based on the above observation. Zero neutral, one right, two left.
- verticalMovement: The output vertical action made by the player based on the above observation. Zero neutral, one up, two down.
- flash: The output flash action made by the player based on the above observation. Zero neutral, one flash.
- taunt: The output taunt action made by the player based on the above observation. Zero neutral, one taunt.

This data was inputted into the ANN to obtain a prediction. This prediction was then compared to the real action that the human did.

E. Human assessment tests

Previously, it was explained that a human-test was required to assess the quality of the agent. For this test, a query was created for the cloned subjects. The query consisted of 4 match videos. In the matches there were four clones. One clone was the subject clone, another one was a control agent that trained just 10 iterations so it can be considered a random clone, the remaining two agents were the other subjects clones selected randomly. The first video was a training video, all agents inside the video were identified. For the next three videos the agents starting position and colors were switched. The query presented then 3 questions.

- What color is your clone?
  - Red
  - Blue
  - Green
  - Yellow
- How confident is your choice?
  - Strongly Agree
  - Agree
  - Partially Agree
  - Disagree
- Why did you choose that clone?

The first two questions were multi-choice and the last one was an open-ended question. This query was made to understand if the clone's owner was capable of differentiating between the others thus implying there are subjective differences between the clones. Being four clones the possibility of choosing randomly and matching the correct clones still reasonable. For that reason the confidence and reasoning behind the selection was asked. The results of this query are presented in the next chapter.

V. Imitation quality results

For the tests, 6 agents were trained by different human subjects, the agents were code named **AR, BH, MF, TS, VC and VG**. A seventh agent code named **EF** is the agent trained by this project main researcher. The first set of test were done by inputting previously recorded games into each of the 7 agents. These games were all played and recorded by this project main researcher as reference. The objective of this test was to check whether a clone trained by a person A was capable of predicting person B game. If clones are truly different they should have different predictions, because of the different play-style of each person.

A. Confusion matrices for flash and taunt

We will begin with the flash and taunt action found in 7 and 8. The graphs show differences between each clone showcasing that some clones are better predictors. Nonetheless, the graph for these actions are very spread and chaotic as shown in 7 and 8. In the flash graph the **EF** clone is one of the best predictors having the highest average and peaks but with medians close to **AR**. Both the flash and taunt graph validate the idea that clones are indeed different, However the idea that the clone of one person is the best at predicting that person game may be challenged if we check the taunt graph.

For the taunt case **EF** predicting **EF** is one of the worst predictions. It is important to notice that the taunt action is not the most consistent action as shown in 8 the taunt values are very spread. So the explanation for this result where **MF** is predicting **EF** is the same as before, the **MF** is trying to taunt much more than the **EF** brain. This explanation can be further supported by the **MF**'s accuracy which is lower than **EF**'s, the best values for accuracy in **MF** are the worst for **EF** as seen in 8.

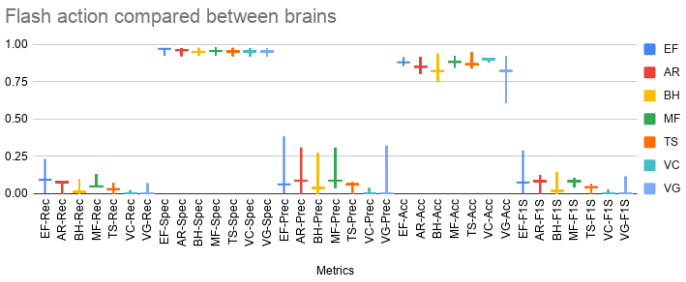

Fig. 7. Flash action by brains box plot.

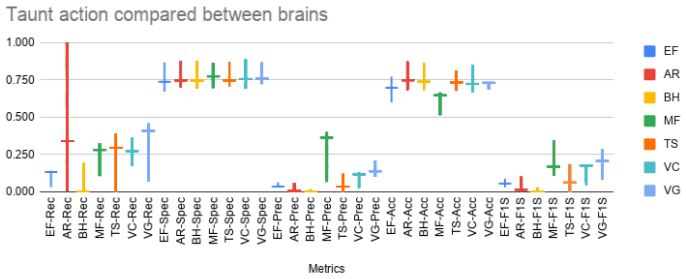

Fig. 8. Taunt action by brains box plot.

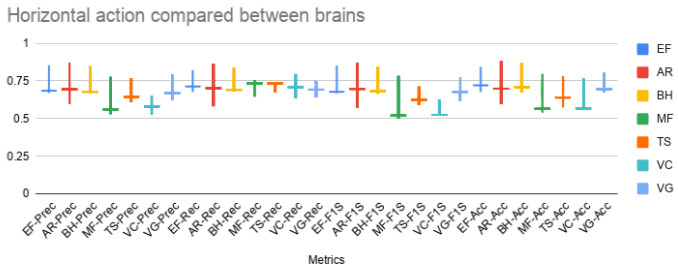

Fig. 9. Horizontal action by brains box plot.

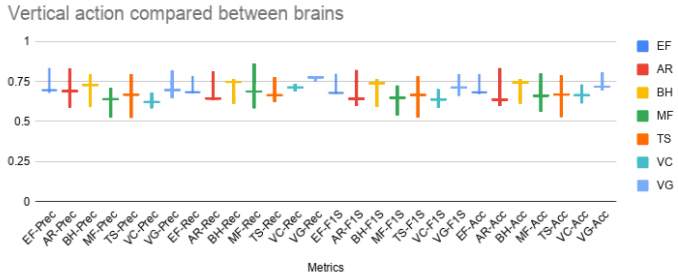

Fig. 10. Vertical action by brains box plot.

B. Confusion matrices for navigation

For the navigation, the comparisons are shown in 9 and 10. The values are close between each agent, near or beyond the 0.70 mark. But there are clear differences between each other. This supports the idea that each clone is different. For the idea that the clone of the person is the best at predicting that person action. **EF** predicting **EF** is one of the best performing brain as shown by its average being among the highest, However is surpassed by **AR**, **BH**, **MF** in some peaks as shown in 9 and 10. Nonetheless, **EF**'s values are more consistent than **AR**, **BH** and **MF**. Meaning that while other brains are able to predict **EF** their values are more spread supporting the idea that **EF** is the best predictor of **EF**. However, It is important to remark that overall differences are not big enough and **EF** is barely ahead of others. Suggesting that while some brains might be better predictors than others it is difficult to differentiate between each clone navigation pattern. The navigation patterns similarities could be explained by the influences coming from the map layout, the map is static and has more width than height, forcing the players to move horizontally most of the time to reach their objectives.

C. Global analysis parameters

The global metrics comparisons can be seen in 11. For this case, **BH** is the best brain, compared to **EF** in all metrics. Both hamming loss and imperfect matches are lower than **EF**. The **BH** brain was one of the closest to the **EF** brain in the previous actions. So it is no surprise that it is able to make good predictions. Nonetheless, the difference between **EF** and **BH** prediction is not big.

D. Survey recognition results

In 12, the total results of the queries can be seen. There were a total of 18 recognition, 3 games each person multiplied by 6 subjects. Of those 18, 10 were correct and 8 were incorrect. Meaning that barely more than half of the recognition attempts were correct. However, this value is better than if answers were chosen randomly which would give a 25% chance of answering correctly or 33% if the control agent is discounted. This may seem to validate the idea than the agents are indeed playing differently or at least there is something that can be recognized. However, being so close also means that the difference is not enough to be distinguished consistently.

E. Recognition confidence

In the survey, the following affirmation was present: "I am confident in my choice", the subjects could answer either, "Strongly Agree", "Agree", "Partially Agree", "Disagree". The idea behind this was to understand the confidence of the subjects while answering. This is because they only have 4 choices, meaning that they had a certain probability of answering correctly by chance. The graph in 13 shows that the subjects were in doubt. And that doubt was reflected in the recognition attempt because both "Agree" and "Partially Agree" successful and failed recognition are equal. However, only one of the subjects completely disagreed and that subject successfully identified the clone. Which shows that even in doubt the subjects perceived some differences or quirks in the clones that helped them differentiate between the clones, but not enough to for them to be sure.

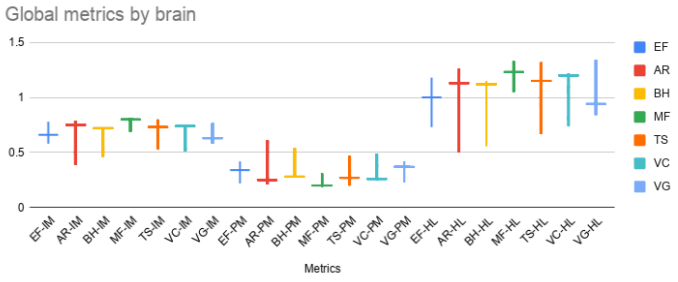

Fig. 11. Global metrics by brain box plot.

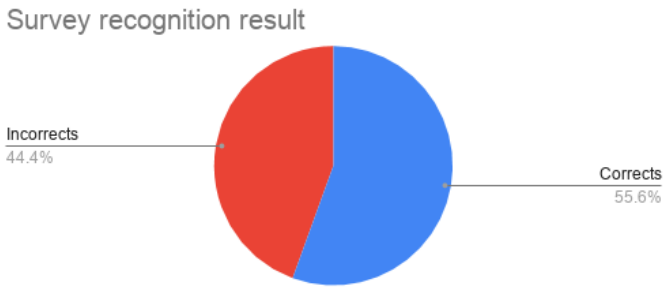

Fig. 12. Survey recognition result.

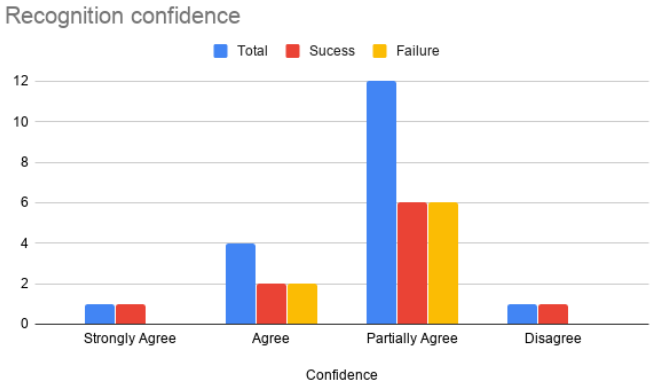

Fig. 13. Survey confidence result.

VI. Conclusions

This project sought to solve the problem of creating an artificial player for Pic-a-boo through the use of an ANN. The idea was that an ANN could be trained to produce actions similar to a human. The ANN would be trained using **symbolic** observations of the game environment for it to produce actions the same way a human would do.

This project manages to do that thanks to a three entity architecture **Academy, Brain and Agent** alongside a feed-forward ANN, that received observations of the state of the games and sent the actions to perform to each agent. A simplified environment was created to speed-up the development time avoiding heavy modifications into the already implemented game. This environment called allowed the training of several agents that were created to imitate six different subjects. The AI algorithm used to create these clone was a feed-forward ANN created by using the API provided by UMLAT.

To assess the quality of the resulting agent, this project used one of the most common tools to evaluate ANN classifiers, the confusion matrix. The confusion matrix allowed the measurement of the agents after training to check their progress, to compare agents between subjects and to choose the best configurations for the clones. In general, the resulting agents are acceptable in their play style, the strong point of the agent is navigation which is a big part of what Pic-a-boo requires. However, the capacity for flashing and taunting is quite limited because these events are not as common in the game and so the agents are not able to learn these actions. Another weak point of the agent is to understand the sequence of flashing which means it doesn't try to flash the targets in the sequence required to win, instead it favors the nearest target. This could be solved by using a recurrent ANN, which was one of the desired options to explore but UMLAT it is still in beta and the functionality at the time of writing was bugged. Nonetheless, the resulting clones are still tolerable and provide a good challenge for players.

In addition to the previously mentioned weak points, cloning different subjects lead to similar results. Even the cloned subjects had a hard time recognizing their own clones when asked to identify them. This fact was confirmed when the clone of subject A was used to predict subject B game trace. The clone of subject A was able to predict a great part of subject B game, nonetheless, the subject A clone is normally the best predictor of subject A game.

References

[1] D. L. Georgios N. Yannakakis, Pieter Spronck, "Player modeling," 2013.

[2] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Personas versus clones for player decision modeling," in *ICEC*, 2014.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[4] S. F. Gudmundsson, P. Eisen, E. Poromaa, A. Nodet, S. Purmonen, B. Kozakowski, R. Meurling, and L. Cao, "Human-like playtesting with deep learning," *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, 2018.

[5] K. Kunanusont, S. M. Lucas, and D. P. Liebana, "General video game ai: Learning from screen capture," *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2078–2085, 2017.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents (extended abstract)," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[7] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," *Journal of Artificial Intelligence Research*, 2016.

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *2012 IEEE/RSJ International Conference*, 2012.

[9] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, 2018.

[10] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *ArXiv*, vol. abs/1809.02627, 2018.

[11] Tensorflow, "Tensorflow: Open source machine learning," 2019.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint*, 2017.
[13] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," *arXiv preprint*, 2017.
[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.
[15] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, pp. 21:1–21:35, Apr. 2017.
[16] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying mcts for human-like general video game playing," in *IJCAI*, 2016.
[17] D. Proudfoot, "Anthropomorphism and ai: Turing s much misunderstood imitation game," *ScienceDirect*, 2011.
[18] J. Patterson and A. Gibson, *Deep Learning A Practitioner's Approach*. USA, 1005 Gravenstein Highway North,Sebastopol, CA 95472: O'Reilly, 1 ed., 2017.
[19] B. Shmueli, "Multi-class metrics made simple, part i: Precision and recall," 2019.
[20] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "A survey on multi-output learning," *arXiv preprint*, 2019.
[21] U. Technologies, "Training config files," 2019.
[22] C. Bishop, *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.
[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, 2015.